

# U5 Fortgeschrittene AVR-Programmierung

---

- Interrupts
- volatile-Variablen
- Synchronisation mit Unterbrechungsbehandlungen
- Stromsparmodi des AVR

# U5-1 Externe Interrupts des AVR- $\mu$ C

## 1 Flanken-/Pegel-Steuerung

- Externe Interrupts durch Pegeländerung an bestimmten I/O-Pins
  - ◆ ATmega8: 2 Quellen an den Pins PD2 und PD3
- Für jeden Interrupt kann bestimmt werden, ob er pegel- oder flanken-gesteuert ist
  - Steuerung für Interrupt  $n$  durch Interrupt Sense Control Bits ( $ISCn0$  und  $ISCn1$ )

| <b>ISCn1</b> | <b>ISCn0</b> | <b>IRQ bei:</b>          |
|--------------|--------------|--------------------------|
| <b>0</b>     | <b>0</b>     | <b>low-Pegel</b>         |
| <b>0</b>     | <b>1</b>     | <b>jedem Wechsel</b>     |
| <b>1</b>     | <b>0</b>     | <b>fallender Flanke</b>  |
| <b>1</b>     | <b>1</b>     | <b>steigender Flanke</b> |

# 1 Flanken-/Pegel-Steuerung (2)

- Beim ATmega8 befinden sich die ISC-Bits im MCU Control Register (MCUCR)
- Position der ISC-Bits in den Registern durch Makros definiert  $ISCn0$  und  $ISCn1$
- Beispiel:  $INT0$  bei ATmega8 für fallende Flanke konfigurieren

```
/* die ISCs für INT0 befinden sich im MCUCR */  
MCUCR &= ~(1<<ISC00);      /* ISC00 löschen */  
MCUCR |= (1<<ISC01);       /* ISC01 setzen */
```

## 2 Maskieren

- Alle Interruptquellen können separat ausgeschaltet (=maskiert) werden
- Für externe Interrupts ist folgendes Register zuständig:
  - ◆ ATmega8: General Interrupt Control Register (GICR)
- Die Bitpositionen in diesem Register sind durch Makros `INTn` definiert
- Ein gesetztes Bit aktiviert den jeweiligen Interrupt
- Beispiel: Interrupt 0 aktivieren

```
GICR |= (1<<INT0);      /* demaskiere Interrupt 0 */
```

## 2 Maskieren (2)

- Alle Interrupts können nochmals bei der CPU direkt abgeschaltet werden
  - durch speziellen Maschinenbefehl `cli`

- Die Bibliothek `avr-libc` bietet hierfür Funktionen an:  
(`#include <avr/interrupt.h>`)

- ◆ `sei()` - lässt Interrupts zu
- ◆ `cli()` - blockiert alle Interrupts

- Beispiel

```
#include <avr/interrupt.h>

sei();                      /* IRQs zulassen */
```

- Innerhalb eines Interrupt-Handlers sind automatisch alle Interrupts blockiert, beim Verlassen werden sie wieder deblockiert
- Beim Start des  $\mu$ C sind die Interrupts bei der CPU abgeschaltet

### 3 Interrupt-Handler

- Installieren eines Interrupt-Handlers wird durch die verwendete Bibliothek unterstützt
- Makro `ISR` (Interrupt Service Routine) zur Definition einer Handler-Funktion (`#include <avr/interrupt.h>`)
- Als Parameter gibt man dem Makro den gewünschten Vektor an, z. B. `INT0_vect` für den externen Interrupt 0
- Beispiel: Handler für Interrupt 0 implementieren

```
#include <avr/interrupt.h>
static int zaehler;

ISR (INT0_vect) {
    zaehler++;
}
```

- Interruptbehandlung sollte möglichst kurz sein! (warum?)

## U5-2 Das volatile-Schlüsselwort

---

- Manche Variablen werden "von außen" verändert
  - ◆ Hardware-Register
  - ◆ Variablen, auf die mehrere Programmabläufe nebenläufig zugreifen
    - Threads
    - Unterbrechungsbehandlungen
- Kann Probleme bei Compileroptimierungen verursachen

# 1 Beispiel

- Hauptprogramm wartet auf ein Ereignis, das durch einen Interrupt gemeldet wird (dieser setzt **event** auf 1)
- Aktive Warteschleife wartet, bis **event!=0**
- Der Compiler sieht, dass **event** innerhalb der Warteschleife nicht verändert wird
  - der Wert von **event** wird nur einmal **vor** der Warteschleife aus dem Speicher in ein Prozessorregister geladen
  - dann wird nur noch der Wert im Register betrachtet
  - Endlosschleife

```
static char event=0;
ISR (INT0_vect) { event=1; }

void main(void) {
    while(1) {
        while(event == 0) { /* warte auf Event */ }
        /* bearbeite Event */
    }
}
```



## 2 Beispiel (2)

- Lösung: Unterdrücken der Optimierung mit dem **volatile**-Schlüsselwort

```
static volatile char event=0;
ISR (INT0_vect) { event=1; }

void main(void) {
    while(1) {
        while(event == 0) { /* warte auf Event */ }
        /* bearbeite Event */
    }
}
```

- Teilt dem Compiler mit, dass der Wert extern verändert wird
- Wert wird bei jedem Lesezugriff erneut aus dem Speicher geladen

### 3 volatile in einem anderen Zusammenhang...

- Beispiel: Funktion zum aktiven Warten

```
void wait(unsigned int len) {  
    while(len > 0) { len--; }  
}
```

- Die Schleife wird terminieren
- Der Wert von `len` wird bei Verlassen der Funktion verworfen
  - Optimierender Compiler entfernt die komplette Schleife
  - Wartezeit wird stark verkürzt
- Mit **volatile** kann diese Optimierung verhindert werden
  - Sonderfall: `len` wird nicht extern modifiziert

## 4 Verwendung von **volatile**

---

- Fehlendes **volatile** kann zu unerwartetem Programmablauf führen
- Unnötige Verwendung von **volatile** unterbindet Optimierungen des Compilers und führt zu schlechterem Maschinencode
- Korrekte Verwendung von **volatile** ist Aufgabe des Programmierers!

Verwendung von **volatile** so selten wie möglich, aber so oft wie nötig.

## U5-3 Synchronisation mit Interrupt-Handlern

---

- Unterbrechungsbehandlungen führen zu **Nebenläufigkeit**
- Nicht-atomare Modifikation von gemeinsamen Daten
- Kann zu Inkonsistenzen führen
- Einseitige Unterbrechung: Interrupt-Handler überlappt Hauptprogramm
- Lösung: Synchronisationsprimitiven
  - hier: zeitweilige Deaktivierung der Interruptbehandlung

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       |            |             |
| 2           |         |            |             |
| 4           |         |            |             |
| 5           |         |            |             |
| 6           |         |            |             |
| 3           |         |            |             |

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           |         |            |             |
| 4           |         |            |             |
| 5           |         |            |             |
| 6           |         |            |             |
| 3           |         |            |             |

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           | 5       | 4          | -           |
| 4           |         |            |             |
| 5           |         |            |             |
| 6           |         |            |             |
| 3           |         |            |             |

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           | 5       | 4          | -           |
| 4           | 5       | 4          | 5           |
| 5           |         |            |             |
| 6           |         |            |             |
| 3           |         |            |             |



# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           | 5       | 4          | -           |
| 4           | 5       | 4          | 5           |
| 5           | 5       | 4          | 6           |
| 6           |         |            |             |
| 3           |         |            |             |

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           | 5       | 4          | -           |
| 4           | 5       | 4          | 5           |
| 5           | 5       | 4          | 6           |
| 6           | 6       | 4          | 6           |
| 3           |         |            |             |

# 1 Beispiel 1: Lost Update

- Tastendruckzähler: Zählt noch zu bearbeitende Tastendrücke
  - Inkrementierung in der Unterbrechungsbehandlung
  - Dekrementierung im Hauptprogramm zum Start der Verarbeitung

```
/* Hauptprogramm */
volatile unsigned char zaehler;

/* C-Anweisung: zaehler--; */
1 lds r24, zaehler
2 dec r24
3 sts zaehler, r24
```

```
/* Interrupt-Behandlung */

/* C-Anweisung: zaehler++ */
4 lds r24, zaehler
5 inc r24
6 sts zaehler, r24
```

| Instruktion | zaehler | zaehler HP | zaehler INT |
|-------------|---------|------------|-------------|
| 1           | 5       | 5          | -           |
| 2           | 5       | 4          | -           |
| 4           | 5       | 4          | 5           |
| 5           | 5       | 4          | 6           |
| 6           | 6       | 4          | 6           |
| 3           | 4       | 4          | -           |

## 2 Beispiel 2: 16-bit-Zugriffe

### ■ Nebenläufige Nutzung von 16-bit-Werten

```
/* Hauptprogramm */
volatile unsigned int zaehler;

/* C-Anweisung: z=zaehler; */
1 lds r24, zaehler
2 lds r25, zaehler+1
/* z verwenden... */
```

```
/* Interrupt-Behandlung */
/* C-Anweisung: zaehler++ */
3 lds r24, zaehler
4 lds r25, zaehler+1
5 adiw r24,1
6 sts zaehler+1, r25
7 sts zaehler, r24
```

| Instruktion | zaehler | z HP |
|-------------|---------|------|
| 1           | 0x00ff  |      |
| 3-7         |         |      |
| 2           |         |      |

## 2 Beispiel 2: 16-bit-Zugriffe

### ■ Nebenläufige Nutzung von 16-bit-Werten

```
/* Hauptprogramm */
volatile unsigned int zaehler;

/* C-Anweisung: z=zaehler; */
1 lds r24, zaehler
2 lds r25, zaehler+1
/* z verwenden... */
```

```
/* Interrupt-Behandlung */
/* C-Anweisung: zaehler++ */
3 lds r24, zaehler
4 lds r25, zaehler+1
5 adiw r24,1
6 sts zaehler+1, r25
7 sts zaehler, r24
```

| Instruktion | zaehler | z HP   |
|-------------|---------|--------|
| 1           | 0x00ff  | 0x??ff |
| 3-7         |         |        |
| 2           |         |        |

## 2 Beispiel 2: 16-bit-Zugriffe

### ■ Nebenläufige Nutzung von 16-bit-Werten

```
/* Hauptprogramm */
volatile unsigned int zaehler;

/* C-Anweisung: z=zaehler; */
1 lds r24, zaehler
2 lds r25, zaehler+1
/* z verwenden... */
```

```
/* Interrupt-Behandlung */
/* C-Anweisung: zaehler++ */
3 lds r24, zaehler
4 lds r25, zaehler+1
5 adiw r24,1
6 sts zaehler+1, r25
7 sts zaehler, r24
```

| Instruktion | zaehler | z HP   |
|-------------|---------|--------|
| 1           | 0x00ff  | 0x??ff |
| 3-7         | 0x0100  | 0x??ff |
| 2           |         |        |

## 2 Beispiel 2: 16-bit-Zugriffe

### ■ Nebenläufige Nutzung von 16-bit-Werten

```
/* Hauptprogramm */
volatile unsigned int zaehler;

/* C-Anweisung: z=zaehler; */
1 lds r24, zaehler
2 lds r25, zaehler+1
/* z verwenden... */
```

```
/* Interrupt-Behandlung */
/* C-Anweisung: zaehler++ */
3 lds r24, zaehler
4 lds r25, zaehler+1
5 adiw r24,1
6 sts zaehler+1, r25
7 sts zaehler, r24
```

| Instruktion | zaehler | z HP   |
|-------------|---------|--------|
| 1           | 0x00ff  | 0x??ff |
| 3-7         | 0x0100  | 0x??ff |
| 2           | 0x0100  | 0x01ff |

### ■ Weitere Problemszenarien?

### ■ Nebenläufige Zugriffe auf Werte >8-bit müssen i.d.R. geschützt werden!

### 3 Sperren der Unterbrechungsbehandlung beim AVR

---

- Viele weitere Nebenläufigkeitsprobleme möglich
  - ◆ Problemanalyse durch den Anwendungsprogrammierer
  - ◆ Vorsicht bei gemeinsamen Daten nebenläufiger Kontrollflüsse
  - ◆ Auswahl geeigneter Synchronisationsprimitive
- Lösung hier: Einseitiger Ausschluss durch Sperren der Interrupts
  - ◆ Sperrung aller Interrupts (`sei()`, `cli()`)
  - ◆ Maskieren einzelner Interrupts (GICR-Register)
- Problem: Interrupts während der Sperrung gehen evtl. verloren



## U5-4 Stromsparmodi von AVR-Prozessoren

---

- AVR-basierte Geräte oft batteriebetrieben
- Energiesparen kann die Lebensdauer drastisch erhöhen
- AVR-Prozessoren unterstützen unterschiedliche Powersave-Modi
  - Deaktivierung funktionaler Einheiten
  - Unterschiede in der "Tiefe" des Schlafes
  - Nur aktive funktionale Einheiten können die CPU aufwecken
  - Standard-Modus: Idle
- In tieferen Sleep-Modi wird der I/O-Takt deaktiviert
  - nur asynchron arbeitende Einheiten können die CPU aufwecken
  - low-level-gesteuerte externe Interrupts werden asynchron ermittelt
  - flankengesteuerte Interrupts dagegen benötigen den Taktgeber
  - Achtung: Im Simulator mit HapSIM funktionieren pegelgesteuerte Interrupts nicht zuverlässig

# 1 Nutzung der Sleep-Modi

## ■ Unterstützung aus der avr-libc:

`(#include <avr/sleep.h>)`

◆ `sleep_enable()` - aktiviert den Sleep-Modus

◆ `sleep_cpu()` - setzt das Gerät in den Sleep-Modus

◆ `sleep_disable()` - deaktiviert den Sleep-Modus

◆ `set_sleep_mode(uint8_t mode)` - stellt den zu verwendenden Modus ein

## ■ Beispiel

```
#include <avr/sleep.h>
set_sleep_mode(SLEEP_MODE_IDLE); /* Idle-Modus verwenden */
sleep_enable(); /* Sleep-Modus aktivieren */
sleep_cpu(); /* Sleep-Modus betreten */
sleep_disable(); /* Empfohlen: Sleep-Modus danach deaktivieren */
```

## 2 Passives Warten auf Unterbrechungen

- Polling bei passivem Warten nicht möglich (warum?)
- Beispiel:

```
volatile static char event=0;
ISR (INT0_vect) { event=1; }

void main(void) {
    while(1) {

        while(event==0) { /* Warte auf Event */
            sleep_enable();

            sleep_cpu();
            sleep_disable();

        }

        /* bearbeite Event */
    }
}
```

- Synchronisation erforderlich?

## 2 Passives Warten auf Unterbrechungen

- Was passiert, wenn der Interrupt wie unten gezeigt eintrifft?
- Beispiel:


```
volatile static char event=0;
ISR (INT0_vect) { event=1; }

void main(void) {
    while(1) {

        while(event==0) { /* Warte auf Event */
            sleep_enable();
            sleep_cpu();
            sleep_disable();

        }

        /* bearbeite Event */
    }
}
```



## 2 Dornröschenschlaf vermeiden

- Atomarität von Wartebedingungsprüfung und Sleep-Modus-Betreten
- Beispiel:

```
volatile static char event=0;
ISR (INT0_vect) { event=1; }

void main(void) {
    while(1) {
        cli();
        while(event==0) { /* Warte auf Event */
            sleep_enable();
            sei();
            sleep_cpu();          kritischer Abschnitt
            sleep_disable();
            cli();
        }
        sei();
        /* bearbeite Event */
    }
}
```

- **sei** und die Folgeanweisung werden atomar ausgeführt (notwendig?)